

*Note:* Your TA probably will not cover all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. They are deliberately made long so they can serve as a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

## 1 Kazaam

You were at Trader Joe's and you heard a song that was an absolute banger. Sadly, you didn't know what song it was.<sup>1</sup> To help with this, you want to build an app called *Kazaam* that can listen to a small section of a song's lyrics and figure out what song it is.

More specifically, given a string of song lyrics  $L$  and a query lyric string  $Q$ , we want to find if  $Q$  appears as a contiguous substring of  $L$ . As an example, if  $L = \text{'tung tung tung tung sahur'}$ , the answer would be `True` if  $Q = \text{'ahur'}$ , but `False` if  $Q = \text{'tungtung'}$ .

For this problem, we assume that all our strings are made up of ASCII characters, which map each character to a number between 0 and 127 inclusive. The Python function `ord(c)` gives us the ASCII value of character  $c$ .

A simple brute force algorithm solves our problem in  $O(|L||Q|)$  time: there are  $|L| - |Q| + 1$  contiguous substrings of  $L$  that have length  $|Q|$ :

$$\mathcal{L} = \{L_i = (l_i, \dots, l_{|Q|+i-1}) \mid i \in \{0, \dots, |L| - |Q|\}\}.$$

So for each substring  $L_i \in \mathcal{L}$ , check whether  $L_i = Q$ , character-by-character in  $O(|Q|)$  time. If we could determine whether  $L_i = Q$  in  $O(1)$  time instead of  $O(|Q|)$  time, then the above algorithm would run much faster, in  $O(|L|)$  time. We can use hashing to our advantage!

- (a) We start by creating a hash function for our strings. Given some string  $S$ , we want to create a function  $R(S)$  that turns our **string into a number**, such that  $R(S_1) = R(S_2)$  if and only if  $S_1 = S_2$ .

Since we're using ASCII characters, we can actually just represent our string as a  $|S|$ -digit base-128 number. For example, say there are 8 characters in our string, and the characters of  $S$  are represented as  $(s_0, s_1, s_2, \dots)$ . We can compute

$$R(S) = 128^7 s_0 + 128^6 s_1 + 128^5 s_2 + \dots.$$

In this case  $s_0$  is known as the most-significant digit, which means that it gets assigned the highest power of 128.

Describe an algorithm to compute  $R(S)$  using  $O(|S|)$  arithmetic operations, i.e.,  $(-, +, *, //, \%)$ . Note that computing an exponent here is the expensive part: for instance,  $128^8$  will take 8 operations. For parts (a), (b), and (c), assume that one arithmetic operation can operate on integers of arbitrary size.

- (b) Now we have a way to turn a string into a number using  $R(S)$ . Consider  $L_i$ , which is the  $i^{\text{th}}$  substring of our song lyrics  $L$  of size  $|Q|$ . Given  $R(L_i)$ , describe an algorithm to compute  $R(L_{i+1})$  using  $O(1)$  arithmetic operations.

You can assume you have access to a value  $f = 128^{|Q|}$  precomputed.

- (c) Describe an algorithm to check if a query string  $Q$  appears inside a song lyrics string  $L$  using at most  $O(|L|)$  arithmetic operations.

---

<sup>1</sup><https://i.imgur.com/AnUCXDj.png>

The above algorithms pose a problem: if the string  $S$  is large, then  $R(S)$  is roughly  $\Omega(128^{|S|})$ , likely much too large to fit in a single register on your CPU, so we won't be able to perform arithmetic operations on such numbers in  $O(1)$  time. Instead of computing  $R(S)$  directly, let's instead hash it down to a smaller range, specifically

$$R'(S) = R(S) \bmod p,$$

where  $p$  is a randomly chosen large prime number that fits into a machine word so it can be operated on in a CPU register in  $O(1)$  time.

If  $Q = L_i$ , then certainly  $R'(Q) = R'(L_i)$ , and we will be able to identify the match quickly! However, sometimes  $R'(Q) = R'(L_i)$  when  $Q \neq L_i$ , and we will get a false match. Assume that  $p$  can be chosen sufficiently randomly such that false matches are unlikely to occur, i.e., the probability that  $R'(Q) = R'(L_i)$  when  $Q \neq L_i$  is at most  $\frac{1}{|Q|}$ , for any  $L_i \in \mathcal{L}$ .

- (d) Given  $R'(L_i)$  and the value  $f' = 128^{|Q|} \bmod p$ , describe an  $O(1)$ -time algorithm to compute  $R'(L_{i+1})$ .
- (e) Describe an expected  $O(|L|)$ -time algorithm to solve the same problem in part (c) using our new  $R'$ .

## 2 TikTok Audit

TikTok's servers are supposed to store exactly one copy of every video ID in  $[n] = \{1, 2, \dots, n\}$ . We want to audit the stream using much less space than storing all the IDs. Assume every video ID in the stream is an integer in  $[n]$ , and that the stream can only be read once from left to right.

- (a) Suppose the stream has length  $n - 1$  and contains  $n - 1$  distinct video IDs from  $[n]$ . In other words, exactly one video ID from  $[n]$  is missing. Give a one-pass streaming algorithm that outputs the missing ID using  $O(\log n)$  bits of memory.
- (b) Suppose the stream has length  $n$ , but exactly one video ID  $m$  from  $[n]$  is missing, and exactly one video ID  $d$  from  $[n]$  appears twice. Every other video ID appears exactly once. Give a one-pass streaming algorithm that outputs both  $m$  and  $d$  using  $O(\log n)$  bits of memory.